Employing simulation to evaluate designs: The APEX approach

Michael A. Freed, Michael G. Shafto, Roger W. Remington
NASA Ames Research Center
Mail Stop 262-4
Moffett Field, CA 94035-1000
{mfreed,mshafto,rremington}@mail.arc.nasa.gov

1 Iterative design

The enormous cost of fielding a complex human-machine system can be attributed in part to the cost of discovering and eliminating usability problems in its design. In general, evaluation costs increase as the design process progresses. By the time a system has come into use, fixing a design problem involves not only redesigning and re-testing, but also modifying fielded devices and possibly retraining users. To manage engineering design costs, large new systems are usually developed by a process of *iterative design* (Gould, 1988). As a design progresses from idea to fully fielded system, decisions are evaluated at each stage. If problems are discovered during evaluation, the system is partially redesigned and further evaluation takes place on the new version. This process is repeated until a satisfactory version results. Of course, the ability to determine whether the current version is satisfactory is limited by the effectiveness of the evaluation methods employed.

Evaluation methods applicable at a late design stage are generally more expensive but also more effective than methods that can be used at earlier stages. In particular, once a working prototype of the new system has been constructed, evaluation by user testing becomes possible. Observing users employing the system in a wide range of scenarios and operating conditions can tell a designer a great deal about how well it will function once in the field. This process is widely recommended in discussions of human factors and routinely practiced in the design of safety-critical and high-distribution systems.

However, user testing suffers from a number of drawbacks and limitations. For instance, subjects are often more highly motivated than true end-users and, in some cases, become too knowledgeable about the developing system to be useful in discovering certain problems. Another drawback is cost. When designing new air traffic control systems, for example, such tests typically require hiring highly paid expert controllers as subjects, often for extended periods (Shafto, 1990; Remington, 1990b). The limited amount of testing that results from high cost can stifle innovation, slow development, and even compromise safety.

Designers can reduce the amount of user testing required by discovering problems early in the design process, and thus reducing the number of design iterations. To discover problems with usability, the primary early-phase evaluation method involves checking the design against human factors guidelines contained in numerous handbooks developed for that purpose (Smith, 1986). Guidelines have proven useful for some design tasks, but have a number of fairly well-known problems (Mosier, 1986). In particular, guidelines focus on static, relatively superficial factors affecting human-machine performance such as text legibility and color discrimination. But when addressing topics relating to the dynamic behavior of a system or to the mental activities of the user, guidelines are often lacking or are too general to be of much use. Thus, “for the foreseeable future, guidelines should be considered as a collection of suggestions, rather than distilled science or formal requirements. Understanding users, testing, and iterative design are indispensable, costly necessities" (Gould, 1988).

Scenario-based approaches, such as Cognitive Walkthrough (Polson et al., 1992), “thinking aloud,” and human simulation modeling, offer alternative methods for early-stage design evaluation. These techniques trade off some of the guideline-based method's generality for greater sensitivity to human cognitive factors and for an increased ability to predict performance in complex, dynamic task domains. The idea of a scenario-based approach is to achieve some of the benefits of user testing at an early design stage when no usable prototype has been constructed. Designers follow the behavior of a real or hypothetical user employing imaginary or simulated equipment to achieve specified task goals in specified operating conditions.

Method	**when**	**redesign cost**	**method use cost**	**demanding task environments**	**method effectiveness**
User testing	late	high	high	yes	high
Guidelines	early	low	low	no	low
Walkthrough	early	low	low	no	medium
Simulation	early	low	medium	yes	medium

Table 1 Comparison of usability evaluation methods

Focusing on specific scenarios allows designers to consider situation-dependent aspects of performance such as the varying relevance of different performance variables, the effects of changing workload, and the likelihood and consequences of interactions between a user's tasks. However, complexity and dynamic elements in a task domain pose difficulties for any scenario-based approach. While an improvement over guidelines in this respect, all of these approaches become more difficult to use in more demanding task domains as task duration, situation complexity, number of actors, number of activities that each actor must perform, and the number of scenarios that need to be considered all increase.

By exploiting the computer's speed and memory, human simulation modeling overcomes obstacles inherent in other scenario-based methods and thus has the greatest potential for predicting performance in more demanding task environments. A large, accurate memory overcomes the problem of tracking innumerable scenario events. Processing speed helps compensate for the need to examine more scenarios by, in principle, allowing each scenario to be carried out more quickly than in real-time. The computer's ability to function continuously adds further to the number of scenarios that may be explored. However, despite its potential, human simulation has been used to inform design almost exclusively in simple design domains – i.e. domains where tasks are brief, situational complexity is low, few actors and forces determine events, and so on.

Predicting performance in more challenging task domains requires an operator model that can function effectively in demanding task environments. Existing human models have typically lacked several very important capabilities including those needed to cope with varied forms of uncertainty inherent in many task environments; manage limited cognitive, perceptual, and motor resources; and, manage multiple, periodic tasks. These capabilities have been incorporated into a human operator model called APEX (Freed, 1997a; Freed, 1997b) by adapting techniques from the field of artificial intelligence.

APEX has been applied to simulate air traffic controller behavior (ATC), a task domain that presents a variety of challenges for human modeling. Expert performance in this domain requires coping with uncertainty, managing limited resources, and managing multiple tasks – challenges one would expect in many other design domains of practical interest. This paper uses the air traffic control domain to illustrate a five-step process for employing APEX to aid design in a new domain.

1. Constructing a simulated world
2. Task analysis
3. Scenario development
4. Running the Simulation
5. Analyzing simulation results

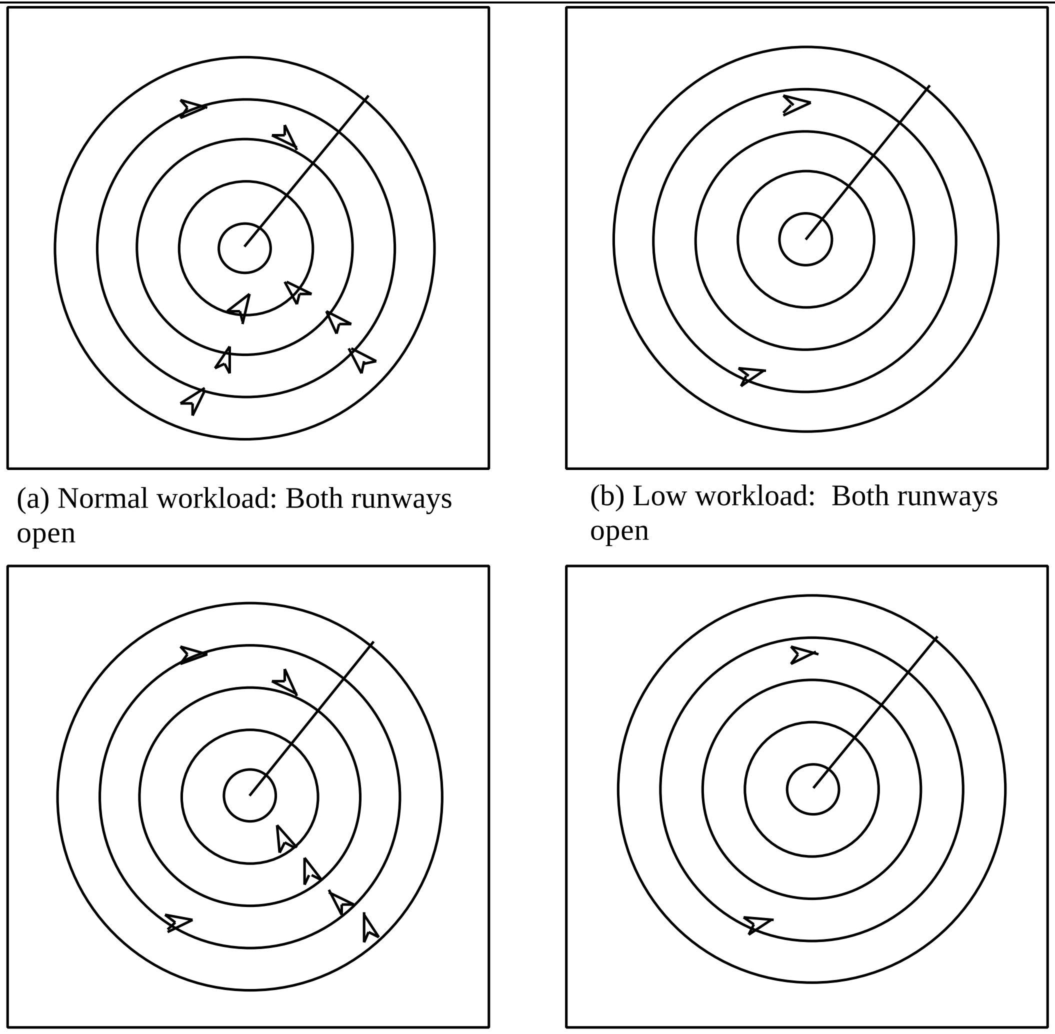
The paper is organized as follows. Section 2 describes a scenario that sometimes occurs in an

APEX ATC simulation; the scenario illustrates how APEX simulation fits into the overall design process and exemplifies its use in predicting operator error. Subsequent sections discuss each of the five steps listed above for preparing and using an APEX model to aid in design.

2 Example scenario

At a TRACON air traffic control facility, one controller will often be assigned to the task of guiding planes through a region of airspace called an arrivals sector. This task involves taking planes from various sector entry points and getting them lined up at a safe distance from one another on landing approach to a particular airport. Some airports have two parallel runways. In such cases, the controller will form planes up into two lines.

Occasionally, a controller will be told that one of the two runways is closed and that all planes on approach to land must be directed to the remaining open runway. A controller's ability to direct planes exclusively to the open runway depends on remembering that the other runway is closed. How does the controller remember this important fact? Normally, the diversion of all inbound planes to the open runway produces an easily perceived reminder. In particular, the controller will detect only a single line of planes on approach to the airport, even though two lines (one to each runway) would normally be expected (see figure 1a and 1b).



(a) Normal workload: Both runways open

(b) Low workload: Both runways open

(c) Normal workload: Left runway

(d) Low workload: Left runway

Figure 1 Radar displays for approach control

However, problems may arise in conditions of low workload. With few planes around, there is no visually distinct line of planes to either runway. Thus, the usual situation in which both runways are available is perceptually indistinguishable from the case of a single closed runway (figure 1c and 1d). The lack of perceptual support would then force the controller to rely on memory alone, thus increasing the chance that the controller will accidentally direct a plane to the closed runway.

Designing to prevent such problems is not especially difficult – it is only necessary to depict the runway closure condition prominently on the controller's information display. The difficulty lies in anticipating the problem. By generating plausible scenarios, some containing operator error, APEX can direct an interface designer's attention to potential usability problems. Though perhaps obvious from hindsight, such errors could easily be overlooked until a late stage of design.

The ability to explicate events (including cognitive events) leading to the error can help indicate alternative ways to refine an interface. For example, one of the difficulties in designing a radar display is balancing the need to present a large volume of information against the need to keep the display uncluttered. In this case, by showing how the error results from low traffic conditions, the model suggests a clever fix for the problem: prominently depict runway closures only in low workload conditions when the need for a reminder is greatest and doing so produces the least clutter.

3 Constructing a simulated world

The first step in simulating a human-machine system involves implementing software components specific to the task domain. Because the domain model used for simulation will almost inevitably require simplifying from the real domain, the exact nature of the tasks the simulated operators will have to carry out cannot be known until this step is accomplished. Constructing software to model the domain thus precedes representing task knowledge for the operator model. This software, the **simulated world**, should include several components:

- ✍✍a model of the **immediate task environment** including **equipment models** specifying the behavior of devices employed by the simulated operator. In ATC, these include a radar scope, two-way radio, and flightstrip board.
- ✍✍a model of the **external environment** specifying objects and agents outside the operator's immediate environment. In ATC, the external environment comprises a region of airspace over which the controller has responsibility, airspace outside that region's boundaries, a set of airplanes, and the aircrews controlling those airplanes.
- ✍✍a **scenario control** component that allows a user to define scenario events (e.g. airliner emergencies, runway closures) and scenario parameters (e.g. plane arrival rate) and then insures that these specifications are met in simulation. See section 5.

In addition, a **simulation engine** controls the passage of simulated time and mediates interactions within and among all simulated world and simulated operator components. A simulation engine provided by the CSS simulation environment, discussed in section 6, is currently used to run the APEX human operator model as well as the air traffic control simulated world described below.

3.1 Air traffic control – a brief overview

APEX has been specified to carry out controller tasks at a simulated terminal radar control (**TRACON**) facility. Controllers at a TRACON manage most of the air traffic within about 30 miles of a major airport. This region is situated within a much larger airspace controlled by an air route traffic control center (ARTCC) – usually just called "Center." TRACON space encompasses small regions of "Tower" airspace, each controlled by a major or satellite airport within the TRACON region. Airspace within a TRACON is normally divided into sectors, each managed by separate controllers. Pilots must obtain controller permission to move from one sector or airspace regime to another.

Controllers and pilots communicate using a two-way radio, with all pilots in a given airspace sector using the same radio frequency. Since only one speaker (controller or pilot) can broadcast over this frequency at a time, messages are kept brief to help control "frequency congestion." Controllers manage events in their airspace primarily by giving **clearances** (authorizations) to pilots over the radio. The most common clearances are:

- ✍✍**handoffs**: clearances that permit a plane to enter one's airspace or, conversely, that tell a pilot about to exit one's airspace to seek permission from the next controller
- ✍✍**altitude clearances**: authorizations to descend or climb. Used at a TRACON mostly to manage takeoffs and landings, but also to maintain safe separation between planes.
- ✍✍**vectors**: i.e. clearances to change heading. The new heading may be specified as an absolute compass direction (e.g. "two seven zero" for East), as a turn relative to the current heading (e.g. "ten degrees left"), or with respect to a named geographical position appearing on navigational charts called a **fix** (e.g. "go direct to DOWNE").
- ✍✍**speed clearances**: authorizations to change airspeed. Managing airspeeds is the most difficult, but in principle the best, way maintain aircraft separation and to space arriving planes for landing.

Clearances are issued according to a standard phraseology (Mills, 1992) to minimize confusion. For example, to clear United Airlines flight 219 for descent to an altitude of 1900 feet, a controller would say, “United two one niner, descend and maintain one thousand nine hundred.” The pilot would then respond with a **readback** – “United two one niner, descending to one thousand nine hundred” – thus confirming to the controller that the clearance was received and heard correctly.

The **radar display** is the controller’s main source of information about current airspace conditions. Each aircraft is represented as an icon whose position on the display corresponds to its location above the Earth’s surface. Planes equipped with a device called a C- or S-mode transponder, including all commercial airliners, cause an alphanumeric **datablock** to be displayed adjacent to the plane icon. Datablocks provide important additional information including altitude, airspeed, airplane type (e.g. 747), and identifying **callsign**. Further information, including the airplane’s planned destination, can be found on paper **flightstrips** located on a “flightstrip board” near the radar display.

As a plane approaches TRACON airspace from a Center region, it appears on the scope as a blinking icon. The controller gives permission for the plane to enter – i.e. accepts a handoff – by positioning a pointer over the icon and then clicking a button. The two-way radio on board the aircraft automatically changes frequency, allowing the pilot to communicate with the new controller. Some planes are not equipped for automatic handoffs, in which case a specific verbal protocol is used:

> *Example*: as a small Cherokee aircraft with callsign 8458R approaches Los Angeles TRACON airspace, the pilot manually changes the radio setting and announces, “LA approach, Cherokee eight four five eight romeo, ten miles north of Pasadena, at four thousand feet, landing.” After detecting the plane on the radar scope, the controller announces “Cherokee eight four five romeo, radar contact,” thereby clearing the plane to operate in LA TRACON airspace.

Standard operating procedures specify nearly every aspect of routine air traffic control at a TRACON, including the time window within which certain clearances should be issued and the flight paths planes should be made to traverse on departure from and landing approach to airports. To continue with the previous example, the following event sequence illustrates a typical (though simplified) landing approach:

- ✍✍After announcing radar contact, the controller locates the Cherokee’s paper flight strip, determines that its destination is Los Angeles International airport (LAX), and selects an appropriate path from the plane’s present position.
- ✍✍The controller vectors the plane along the first leg on this path, saying “Cherokee five eight romeo, cleared direct for DOWNE.” The pilot acknowledges with a readback.
- ✍✍While the plane travels to the DOWNE fix, the controller observes it periodically to insure separation from other aircraft and to determine a safe time to clear it to the correct altitude for the LAX final approach. When appropriate, the controller says, “Cherokee five eight romeo, descend and maintain one thousand nine hundred.”
- ✍✍As the Cherokee approaches DOWNE, the controller selects a preferred runway and then locates a gap in the line of planes approaching that runway. Vectors and speed clearances are used to maneuver it into the gap at safe distance from other aircraft. For example, the plane may need to be 5 miles behind a 747 and 3 miles ahead of whatever follows.
- ✍✍Finally, as the plane nears LAX Tower airspace, the controller initiates a handoff to Tower by saying “Cherokee five eight romeo, cleared for ILS approach. Contact tower at final approach fix.”

3.2 ATC simulation: defining an airspace

A TRACON is typically divided into separate airspace sectors, each handled by one or more individual controllers. The number of sectors usually varies over the course of a day to reflect the amount of expected air traffic. During high-traffic periods, the overall airspace is divided into smaller sectors, thus reducing the number of planes any particular controller needs to handle. For simplicity, the ATC simulation software divides the overall airspace into an **arrivals sector** and a **departures sector**, each handled by a single controller. Examples throughout this document will center on the arrival sector controller at Los Angeles TRACON.

Users can easily define new airspace models in the simulated ATC world. Such models consist of three kinds of objects: airports, fixes, and regions. Defining an airport or fix causes all simulated pilots in the simulated ATC world to know its location; the controller can thus vector planes "direct to" that location. Defining an airport also creates an ATC Tower to which the control of a plane can be handed off. When control of a plane passes to an airport Tower, the plane icon on the simulated radar display disappears soon thereafter.

Regions define operationally significant areas of airspace, possibly but not necessarily corresponding to legal divisions, and not usually encompassed by explicit boundaries on the display. They provide a usefully coarse way to represent plane location, allowing a controller to refer to the area, e.g., "between DOWNE and LAX." The ability to consider airspace regions allows the simulated controller to assess air traffic conditions, facilitates detection of potential separation conflicts, and provides a basis for determining when planes have strayed from the standard flight path. Regions are essentially psychological constructs and are therefore properly part of the agent model, not the domain model. However, regions need to be represented in the same coordinate system as fixes and airports, making it convenient to specify all of them together.

3.3 ATC simulation: controller tasks

In the simulated world, as in the real world, the task of handling an arrival is entirely routine. Most planes arrive from Center space via one of a few pre-established airspace "corridors." The controller periodically checks for new arrivals, represented as blinking plane icons, and then accepts control from center by clicking a mouse button over their icons. Once control of a new plane has been established, the paper flight strip associated with the plane is consulted to determine the flight's planned destination and then marked (or moved) to indicate a change from pending to active status. The simulated world uses "electronic flightstrips" in accordance with somewhat controversial proposals to transfer flightstrip information to the controller's information display (Stein, 1993; Vortac, 1993). The task of routing a plane to its destination – either an airport or a TRACON airspace exit point – proceeds in simulation the same as it does in reality (see example in previous section).

While controllers' tasks are mostly simple and routine when considered in isolation the need to manage multiple tasks presents significant challenge. For instance, the controller cannot focus on one aircraft for its entire passage through TRACON airspace, but must instead interleave effort to handle multiple planes. Similarly, routine scanning of the radar display to maintain awareness of current conditions often must be interrupted to deal with situations discovered during the scanning process, and then later resumed. A further source of challenge is the possibility that certain unusual events may arise and require the controller to adapt routine behavior. For example, if one of the runways at LAX closes unexpectedly, the controller will have to remember to route planes only to the remaining open runway and may have to reduce traffic flow in certain regions to prevent dangerous crowding.

4 Task Analysis

APEX, like other human simulation models, consists of general-purpose components such as eyes, hands, and working memory; it requires the addition of domain-specific knowledge structures to function in any particular task domain. **Task analysis** is the process of identifying and encoding the necessary knowledge (Mentemerlo, 1978; Kirwan, 1992). For highly routinized task domains such as air traffic control, much of the task analysis can be accomplished easily and fairly uncontroversially by reference to published procedures.

For instance, to clear an airplane for descent to a given altitude, a controller uses a specific verbal procedure prescribed in the controller phraseology handbook (see Mills, 1992) – e.g. “United two one niner, descend and maintain flight level nine thousand.” Other behaviors such as maintaining an awareness of current airspace conditions do not correspond to any written procedures. These aspects of task analysis require inferring task representation from domain attributes and general assumptions about adaptive human learning processes. This section introduces the notational formalism (PDL) used in APEX to represent task analyses and discusses the role of adaptive learning in determining how agents come to perform tasks.

4.1 An expressive language for task analyses

In APEX, tasks analyses are represented using the APEX Procedure Definition Language (PDL), the primary element of which is the **procedure**. A procedure in PDL represents an operator's knowledge about how to perform routine tasks. For instance, a procedure for clearing a plane to descend has the following form:

```
(procedure
   (index  (clear-to-descend ?plane ?altitude))
   (step s1 (determine-callsign-for-plane ?plane
          => ?callsign))
   (step s2 (say ?callsign) (waitfor ?s1)
   (step s3 (say "descend and maintain flight
         level") (waitfor ?s2))
   (step s4 (say ?altitude) (waitfor ?s3))
   (step s5 (terminate) (waitfor ?s4)))
```

The **index clause** in the procedure above indicates that the procedure should be retrieved from memory whenever a goal to clear a given plane for descent to a particular altitude becomes active. **Step clauses** prescribe activities that need to be performed to accomplish this. The first step activates a new goal: to determine the identifying callsign for the specified airplane and to make this information available to other steps in the procedure by associating it with the variable ?*callsign*. Achieving this step entails finding a procedure whose index clause matches the form

```
(determine-callsign-for-plane ?plane)
```

and then executing its steps. After this, *say* actions prescribed in steps s2, s3, and s4 are carried out in order. This completes the phrase needed to clear a descent. Finally, step s5 is executed, terminating the procedure.

The activities defined by steps of a PDL procedure are assumed to be concurrently executable. When a particular order is desired, this must be specified explicitly using the **waitfor clause**. In this case, all steps but the first are defined to wait until some other task has terminated. Second, although this task is complete when all of its steps are complete, it is sometimes desirable to allow procedures to specify more complex, variable completion conditions. For example, it may be useful to allow race conditions in which the procedure completes when any of several steps are complete. Thus, rather than handle termination uniformly for all procedures, termination conditions must be notated explicitly in each procedure.

The ability to specify how concurrent execution should be managed and to specialize termination conditions for each procedure exemplify an attempt with PDL to provide a uniquely flexible and expressive language for task analysis. In particular, PDL can be considered an extension to the GOMS approach (Card, 1983) in which tasks are analyzed in terms of four constructs: goals, operators, methods, and selection rules. Procedure structures in PDL combine and extend the functionality provided by GOMS methods and selection rules. GOMS operators represent basic skills such as pressing a button, saying a phrase, or retrieving information from working memory; executing an operator produces action directly. PDL does not produce action directly, but instead sends action requests (signals) to cognitive, perceptual, and motor resources in the APEX resource architecture. What action, if any, should be executed is determined by the relevant resource model.

It is important to distinguish PDL procedures from externally represented procedures such as those that appear in manuals. PDL procedures are internal (cognitive) representations of how to accomplish a task (Anderson, 1995). In some cases, as above, there is a one to one correspondence between the external prescription for accomplishing a task and how it is represented internally. But written procedures might also correspond to multiple PDL procedures, especially when written procedures cover conditional activities (i.e. carried out sometimes but not always) or activities that take place over a long period of time. Similarly, PDL procedures may describe behaviors such as how to scan the radar display that result from adaptive learning processes and are never explicitly taught.

4.2 Approximating adaptive learning

Task analysis is often used to help designers better understand how human operators function in *existing* human-machine systems (Hutchins, 1995). In such cases, task analysis can be usefully (though not altogether accurately) viewed as a linear process in which a task analyst observes operators performing their job, infers underlying cognitive activities based on regularities in overt behavior, and then represents these activities in the context of some general cognitive model.

A different process is required to predict how tasks will be carried out with newly designed equipment and procedures. In particular, analysis can no longer start with observations of overt behavior since no real operators have been trained with the new procedures and no physical realization of the new equipment exists. Instead, cognitive structures underlying behavior must be inferred based on task requirements and an understanding of the forces that shape task-specific cognition: human limitations, adaptive learning processes, and regularities in the task domain.

For example, to model how a controller might visually scan the radar display to maintain awareness of current airspace conditions, an analyst should consider a number of factors. First, human visual processing can only attend to, and thus get information about, a limited portion of the visual field at any one time. By attending to one region of the display, a controller obtains an approximate count of the number of planes in that region. He or she identifies significant plane clusters or other Gestalt groups and can detect planes that differ from all others in the region on some simple visual property such as color or orientation.

But to ascertain other important information requires a narrower focus of attention. For example, to determine that two planes are converging requires attending exclusively to those planes. Similarly, to determine that a plane is nearing a position from which it should be rerouted requires attending to the plane or to the position. These visual processing constraints have important implications for how visual scanning should be modeled. For example, to maintain adequate situation awareness, the model should shift attention not only to display regions but also to individual planes within those regions.

An assumption that the human operator adapts to regularities in the task environment has further implications. For instance, if a certain region contains no routing points and all planes in the region normally travel in a single direction, there would usually be no reason to attend to any particular plane unless it strayed from the standard course. Adaptive mechanisms could modify routine scanning procedures to take advantage of this by eliminating unnecessary attention shifts to planes in that region. This saves the visual attention resource for uses more likely to yield important information.

A fully mature approach to human modeling will require techniques for identifying or predicting regularities in the domain and detailed guidelines for predicting how adaptive learning processes will shape behavior in accordance with these regularities. A few such guidelines have been considered in discussions of particular knowledge representation problems (Freed, 1997a), and somewhat more general principles were discussed as part of APEX's overall modeling methodology (Freed, 1997b). However, the present work has only begun to address this important issue.

5 Scenario development

The third step in preparing an APEX simulation run is to develop scenarios. A scenario specification includes any parameters and conditions required by the simulated world. In general, these can include initial state, domain-specific rate and probability parameters, and specific events to occur over the course of a simulation (see list below). In the current implementation of the simulation, initial conditions do not vary. In particular, the simulated controller always begins the task with an empty airspace (rather than having to take over an active airspace) and with the same set of goals. The goals are to maintain safe separation between all planes, get planes to their destination in a timely fashion, stay aware of current airspace conditions, and so on.

- ✍✍ initial agent goals
- ✍✍ initial operating conditions
- ✍✍ specialized parameters such as the rate and likelihood of certain events

- ✍✍specific events to occur during the simulation run

At minimum, a scenario must include a duration D and an aircraft count C. The scenario control component will randomly generate C plane arrival events over the interval D, with aircraft attributes such as destination, aircraft type, and point of arrival determines according to default probabilities. For instance, the default specifies that a plane's destination will be LAX with p(.7) and Santa Monica airport with p(.3). The default includes conditional probabilities – e.g. the destination airport affects the determination of airplane type – e.g. a small aircraft such as a Cherokee is much more likely to have Santa Monica as its destination.

Scenario definitions can alter these default probabilities, and thereby affect the timing and attributes of events generated by the scenario control component. Users can also specify particular events to occur at particular times. For example, one might want to specify a number of small aircraft arrivals all around the same time in order to check how performance is affected by a sudden increase in workload. Currently, arrivals are the only kind of event that the scenario control component generates randomly. Special events such as runway closures and aircraft equipment failures must be specified individually.

6 Running the Simulation

To employ the operator model, world model, and scenario control elements in simulation requires a simulation engine. APEX currently uses the simulation engine provided by CSS (Remington, et al., 1990a), a simulation package developed at NASA Ames that also includes a model development environment, a graphical interface for observing an ongoing simulation, and mechanisms for analyzing and graphing temporal data from a simulation run.

CSS simulation models consist of a network of **process** and **store** modules, each depicted as a "box" on the graphical interface. Stores are simply repositories for information, though they may be used to represent complex systems such as human working memory. Process modules, as the name implies, cause inputs to be processed and outputs to be produced. A process has five attributes: (1) a name; (2) a body of LISP code that defines how inputs are mapped to outputs; (3) a set of stores from which it takes input; (4) a set of stores to which it provides output; and (5) a stochastic function that determines its finishing time – how much simulated time is required for new inputs to be processed and the result returned as output. A process is idle until a state change occurs in any of its input stores. This activates the process, causing it to produce output in accordance with its embedded code after an interval determined by its characteristic finishing time function.

The CSS simulation engine is an event-driven simulator. Unlike time-slice simulators, which advance simulated time by a fixed increment, an event-driven simulator advances until the next active process is scheduled to produce an output. This more efficient method makes it practical to model systems whose components need to be simulated at vastly different levels of temporal granularity. In particular, the APEX human operator model contains perceptual processes that occur over tens of milliseconds, motor and cognitive processes that take hundreds of milliseconds and (links to) external simulated world processes modeled at the relatively coarse temporal granularity of seconds. CSS provides further flexibility by allowing processes to run concurrently unless constrained to run in sequence.

The process of incorporating a model into a CSS framework is fairly straightforward, but a user must decide how much detail to include regarding the model's temporal characteristics. In the simplest case, one could model the world and the operator each as single processes. Because processes can have only a single finishing time distribution, such a model would assume a uniform duration for all operator activities. For instance, a speech act, a gaze shift, a grasp action, and a retrieval from memory would all require the same interval. The process-store network used to simulate and visualize APEX behavior models each component of the APEX architecture as a separate process.

Once the process-store network has been constructed, and simulated world and APEX elements incorporated into the code underlying processes, the simulation can be run. CSS provides a "control panel" window with several buttons. SHOW toggles the visualization function, causing information in processes and stores to be displayed and dynamically updated. START and STOP initiate and freeze a simulation run. STEP causes time to advance to the next scheduled process finish event, runs the scheduled process, and then freezes the simulation.

7 Simulation analysis

The final step in using APEX is to analyze the results of simulation. CSS provides tools for analyzing and graphing temporal aspects of behavior. For example, if interested in predicting how much time the controller took to respond when a new airplane appeared on the radar display, the modeler could specify that interest when constructing the process-store network (see Remington et al., 1990a for how this is accomplished). CSS automatically stores specified timing values from multiple simulation runs and graphs the data on demand.

7.1 Design-facilitated errors

APEX is intended to help predict **design-facilitated errors** – i.e. operator errors that could be prevented or minimized by modifying equipment or procedure design. The current approach assumes that people develop predictable strategies for circumventing their innate limitations and that these strategies make people prone to error in certain predictable circumstances. For instance, to compensate for limited memory, people sometimes learn to rely on features of their task environment to act as a kind of externalized memory. If, for whatever reason, the relied on feature is absent when it should be present (or vice-versa), error may result.

In the *wrong runway scenario* described in section 2, the controller's error stemmed from reliance on a visually observable feature – an imbalance in the number of planes approaching to each runway to signal – to act as a reminder of runway closure. When workload dropped too low for this feature to remain observable, the controller reverted to a behavior consistent with its absence. In particular, the controller selected a runway based on factors that had nothing to do with runway availability such as airplane type and relative proximity to each runway approach path.

When this error occurs in simulation, the sequence of events that led up to it can be extracted from the **simulation trace**, a record of all the events that occurred during the simulation run. However, this "raw" event data is not very useful to a designer. To inform the design process, the events must be interpreted in light of general knowledge about human performance. For instance, most errors can be partially attributed to the non-occurrence of normal events. The raw simulation data will not contain any reference to these events, so normative knowledge must be used to complete the causal story that explains the error.

As an additional constraint on what constitutes a useful analysis, the explanation for an error assign blame to something that the designer has control (Owens, 1991, makes a similar point). For instance, citing human memory limitations as a cause of the above described error is correct, but not very useful. In contrast, blaming the failure on the absence of expected perceptual support for memory implies ways of fixing the problem. The designer could enforce the perceptual support (in this instance, by insuring that planeload never drops too low), provide alternative perceptual support (a runway closure indicator on the display), or train the operator not to expect perceptual support and to take other measures to support memory.

7.2 Error patterns

We would like to facilitate the generation of analyses in which non-occurring normal events are made explicit and causal explanations trace back to elements of the task environment that the designer might be able to control. One way to do this is to represent general knowledge about the cause of error in **error patterns**. An error pattern is a specific type of explanation pattern (Schank, 1986) – i.e. a stereotypical sequence of events that end in some kind of anomaly that needs to be explained (an error in this case). When an error occurs in simulation, error patterns whose characteristic anomaly type matches the "observed" error are compared against events in the simulation trace. If the pattern matches events in the simulation trace, the patter is considered an explanation of the error.

Error patterns derive from a general theory of what causes error. To make the idea of error patterns concrete, the example below describes a simpler form of error that the one described in section 2.

Because an APEX human operator model can only be a coarse approximation of a real human operator, error predictions emerging from simulation will not necessarily be problems in reality. The designer must evaluate the plausibility and seriousness of any error predictions on the basis of domain knowledge and a common sense understanding of human behavior. Current scientific knowledge about human error-making is inadequate for prediction. The APEX approach only attempts to make designers more effective at applying their common sense knowledge about when and why people make errors. Thus, the need for the user to evaluate model predictions should be considered compatible with the APEX approach.

One other aspect of simulation analysis presents more of a problem. Currently, the modeler must interpret simulation event data "by hand" on the basis of informally specified error pattern knowledge. This approach is far from ideal and, given the massive mount of simulation data that must be examined, probably unacceptable for practical use. To automate analysis, simulation mechanisms must be augmented to check observed (i.e. simulated) behavior against expected behavior and to signal errors when specified deviations occur. Error patterns indexed by the anomaly and successfully matched against the simulation trace would then be output to the user as error predictions.

8 Conclusion

In safety-critical domains, such as nuclear power, aerospace, military, medical, and industrial control systems, the cost and risk of implementing new technology are major barriers to progress. The fear of innovation is not based on superstition, but on the common experience of failure in complex system development projects (Curtis et al., 1988). Retaining the status quo, however, becomes less and less tenable as existing systems become obsolete and the cost and risk of maintaining them escalate.

Replacement or significant upgrading of such safety-critical systems eventually becomes inevitable. Therefore, it is necessary to attack the core problem, namely, the lack of a systematic design method for complex human-computer systems. It is the absence of such a methodology that lies at the root of valid concerns about the safety (Leveson, 1995) and economic benefit (Landauer, 1995) of new human-computer systems.

APEX is intended to be a contribution toward improving the design of safety-critical human-computer systems, for example, the next-generation air traffic control system. APEX incorporates and extends many of the functional elements of the MIDAS system (Corker, 1993, 1995), particularly computational models of the physical and informational environment: equipment, geography, regulatory constraints, documents, and displays.

The key innovations of APEX are its integrated approaches to task analysis, procedure definition, and intelligent, resource-constrained multi-tasking. This paper has presented a step-by-step description of how APEX is used, from scenario development through trace analysis.

The development of APEX is itself an exercise in iterative design. Current work is aimed at extending the modeling framework, developing new applications, and validating partial models (see Homer, 1997). The goal remains to reduce the cost and risk of the implementation of complex human-computer systems, by addressing key human-interaction issues as early as possible in the design process.

Example:

The UNIX command **rm** normally deletes files, thus freeing up hard drive space, but can be redefined to move files into a “trash” directory instead. A user unaware that rm has been redefined may try to use it to make space on the disk for a new file. After typing the command, the system returns a new command line prompt but does not describe the outcome of the command just given. As this response was the expected result of a successful delete, the user believes that the action succeeded in freeing up disk space.

The following general causal sequence constitutes an error pattern that can be specified to match the above incident:

TIME-1	**TIME-2**	**TIME-3**
agent wants G, achievable by action A with prereq P	agent does B and observes E	agent does A which fails to achieve G
agent believes not(P)	agent believes P achieved	
agent believes action B results in P and observable event E	agent believes A will achieve G	
B actually results in Q ($\neq$P) which produces event E		

G	=	get non-local file
A	=	ftp <file-name>..
B	=	rm <file-name>
P	=	disk space available
Q	=	disk space unchaned
E	=	command line prompt

9 References

[Anderson, 1995] Anderson, J.R. (1995) *Cognitive psychology and its implications* (fourth edition), San Francisco: W.H. Freeman.

[Card, 1983] Card, S.K., Moran, T.P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

[Corker, 1993] Corker, K.M., & Smith, B. (1993). *An architecture and model for cognitive engineering simulation analysis: Application to advanced aviation analysis.* AIAA Conference on Computing in Aerospace. San Diego, CA.

[Corker, 1995] Corker, K.M. & Pisanich, G.M. (1995). Analysis and modeling of flight crew performance in automated air traffic management systems. *Proceedings of the 6th IFAC/IFIP/IFORS/IEA Symposium: Analysis, Design, and Evaluation of Man-Machine Systems*. Boston, MA.

[Curtis et al., 1988] Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, **31**, 1268-1287.

[Freed, 1997a] Freed, M.A. & Remington, R.W. (1997). Managing decision resources in plan execution. In *Proceedings of the Fifteenth Joint Conference on Artificial Intelligence*. Nagoya, Japan.

[Freed, 1997b] Freed, M. & Shafto, M. (1997). Human-system modeling: some principles and a pragmatic approach. *Proceedings of the Fourth International Workshop on the Design, Specification, and Verification of Interactive System.* Granada, Spain.

[Gould, 1988] Gould, J.D. (1988). How to design usable systems. In M. Helander (Ed.), *Handbook of Human-Computer Interaction*. New York: North-Holland.

[Homer, 1997] Homer, J.B. (1997). Structure, data and compelling conclusions: Notes from the field. *System Dynamics Review*, **13**, 293-309.

[Hutchins, 1995] Hutchins, E. (1995). *Cognition in the wild*. Cambridge, MA: MIT Press.

[Kirwan, 1992] Kirwan, B. and Ainsworth, L. (1992). *A guide to task analysis.* London: Taylor and Francis.

[Landauer, 1995] Landauer, T.K. (1995). *The trouble with computers*. Cambridge, MA: MIT Press (Bradford).

[Leveson, 1995] Leveson, N.G. (1995). *Safeware: System safety and computers*. Reading, MA: Addison-Wesley.

[Mentemerlo, 1978] Mentemerlo, M.D. and Eddowes, E. (1978). The judgmental nature of task analysis. In *Proceedings of the Human Factors Society,* pp. 247-250. Santa Monica, CA.

[Mills, 1992] Mills, T.S. & Archibald, J.S. (1992). *The pilot's reference to ATC procedures and terminology*. Van Nuys, CA: Reavco Publishing.

[Mosier, 1986] Mosier, J.N. & Smith, S.L. (1986). Applications of guidelines for designing user interface software. *Behavior and Information Technology,* **5**, 39-46, 1986.

[Owens, 1991] Owens, C. (1991). A functional taxonomy of abstract plan failures. In *Proceedings of the Annual Conference of the Cognitive Science Society*. Chicago, IL.

[Polson et al., 1992] Polson, P., Lewis, C., Rieman, J., Wharton, C., and Wilde, N. (1992). Cognitive Walkthroughs: A method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies*, **36**, 741-773.

[Remington, et al., 1990a] Remington, R.W., Johnston, J.C., Bunzo, M.S., & Benjamin, K.A. The Cognition Simulation System: An interactive graphical tool for modeling human cognitive processing. In *Object-Oriented Simulation*. San Diego: Society for Computer Simulation, pp. 155-166, 1990.

[Remington, 1990b] Remington, R.W., & Shafto, M.G. (1990). *Building human interfaces* *to fault diagnostic expert systems I: Designing the human interface to support cooperative fault diagnosis.* Seattle, WA: CHI'90 Workshop on Computer-Human Interaction in Aerospace Systems.

[Schank, 1986] Schank, R.C. (1986). *Explanation patterns*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Shafto, 1990] Shafto, M.G., & Remington, R.W. (1990). *Building human interfaces to fault diagnostic expert systems II: Interface development for a complex, real-time system.* Seattle, WA: CHI'90 Workshop on Computer-Human Interaction in Aerospace Systems.

[Smith, 1986] Smith, S.L., & Mosier, J.N. (1986). *Guidelines for designing operator interface software*. Tech. Rept. No. MTR-10090. McClean, VA: MITRE Corporation.

[Stein, 1993] Stein, E.S. & Garland, D. (1993). *Air traffic controller working memory: considerations in air traffic control tactical operations*. FAA technical report DOT/FAA/CT-TN93/37, 1993.

[Vortac, 1993] Vortac, O.U., Edwards, M.B., & Fuller, D.K. (1993). Automation and cognition in air traffic control: An empirical investigation. *Applied Cognitive Psychology*, **7**, 631-651.